PRM2T

Raport z Etapu II

-

Temat - Kakuro

Bartosz Dorobek, Mikołaj Król, Łukasz Świątek

30 kwietnia 2020

Stworzyliśmy plik easy\_1.csv przechowujący przykładową planszę na najprostszym poziomie. Separatorem w pliku jest przecinek. W pierwszym przechowywane wymiary danej planszy: liczba wierszy x liczba kolumn. Całą planszę opisują 4 tablice o czterech typach w zależności od typu komórek, który zwierają. Tablicę „types” określającą typy wszystkich komórek:

x – komórka pusta

c – suma wartości w kolumnie

r – suma wartości w wierszu

v – wartości

Komórki typu „v” będą na początku puste, a użytkownik będzie mógł je uzupełniać. Program zna jednak ich prawidłowe uzupełnienie. Na podstawie tego będzie mógł je wyświetlać, jeśli użytkownik zażąda podpowiedzi lub rozwiązania planszy. Porównując wartość wpisaną przez użytkownika z faktyczną wartością komórki „v” będzie mógł powiedzieć graczowi czy popełnił błąd. W planszach na wyższych poziomach trudności pojawiać się będą także komórki przechowującą wartość z typu „c” i „r”. Poniżej plansza którą przedstawia plik easy\_1.csv.

Obraz zawierający ekran, zegar, zamontowane, biały

Opis wygenerowany automatycznie

Napisaliśmy program który wczytuje plik z planszami, odczytuje je i zapisuje w macierzach. Plansza „types” jest przechowywana w macierzy typu String[][], a pozostałe w macierzach typu int[][]. Format plików przechowujący poszczególne planszy zawsze będzie taki sam. Pierwszy wiersz dwie wartości określające wymiary planszy. Kolejne typy oddzielone jednym wierszem przeznaczonym na opis słowny danego typu dla czytelności. Program znając wymiary może w dalszej kolejności przypisać odpowiednią liczbę wierszy z pliku do macierzy różnych typów.

Plik easy\_1.csv

5,5

--------------types

x,c,c,x,x

r,v,v,c,x

r,v,v,v,x

x,r,v,v,v

x,x,r,v,v

--------------values

0,0,0,0,0

0,1,3,0,0

0,2,1,3,0

0,0,5,1,6

0,0,0,4,7

--------------columns

0,3,9,0,0

0,0,0,8,0

0,0,0,0,13

0,0,0,0,0

0,0,0,0,0

--------------rows

0,0,0,0,0

4,0,0,0,0

6,0,0,0,0

0,12,0,0,0

0,0,11,0,0

W dalszej kolejności chcemy by na podstawie tych macierzy program mógł zapisać ich wartości do klas reprezentujących odpowiednie rodzaje komórek Cell. W tych klasach pojawią się funkcję umożliwiające użytkownikowi wpisywanie do nich wartości i ich usuwanie i realizujące inne funkcjonalności opisane w etapie pierwszym . Chcemy stworzyć w JavaFx GUI, które będzie korzystać z odpowiednich klas oraz funkcji i wyświetli interaktywną planszę.

Podsumowując nasz plan wykorzystanych klas:

- GUI - będzie to pewnego rodzaju interfejs, w którym użytkownik będzie wybierać, co chce wykonać tzn. wczytanie gry, rozpoczęcie nowej gry, wyjście z gry,

- BoardGenerator - klasa odtwarzająca planszę do gry (z wybranego poziomu) z pliku .csv wcześniej przez nas przygotowanego (plansza w pliku jest już uzupełniona, ale użytkownikowi pokazuje się pusta w polach do uzupełnienia),

- Checker, Solver - klasa sprawdzająca czy plansza jest uzupełniona poprawnie porównuje są z planszą zapisaną w pliku,

- Saver - klasa służąca do zapisywania aktualnego stanu gry do pliku .csv,

- Loader - klasa wczytująca poprzednią grę z pliku wcześniej zapisanego,

- Cell - klasa komórki, która będzie dziedziczyła kilka podrodzajów komórek tj.

* EmptyCell (x) – pusta komórka (nie wchodzi w żadną interakcję),
* ColumnCell (c) – suma wartości w kolumnie,
* RowCell (r) – suma wartości w wierszu,
* ValueCell (v) - komórka, którą ma uzupełnić użytkownik.

Przykładowe algorytmy wykorzystane w klasach:

- Check - Sprawdzenie czy wpisana odpowiedź w komórce jest poprawna,

- Save - zapisanie aktualnej gry do pliku .csv,

- GenerateBoard - wczytanie planszy gry,

- Hint - odsłonienie losowej niezapisane komórki,

- Load - odtworzenie zapisanego stanu gry,

- Solve - ujawnia poprawne odpowiedzi.

package com.kakuro;  
  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Arrays;  
import java.util.Scanner;  
  
public class Main {  
 */\*\* funkcja wypisująca macierz w formie kolejnych rzędów\*/* public static void print2D(int[][] mat)  
 {  
 // Loop through all rows  
 for (int[] row : mat)  
  
 // converting each row as string  
 // and then printing in a separate line  
 System.*out*.println(Arrays.*toString*(row));  
 }  
 */\*\* funkcja odczytująca z pliku macierz typu String\*/* private static String[][] readBoardString(Scanner inputStream, int rows\_number, int columns\_number) {  
 inputStream.next();  
 String[][] data = new String[rows\_number][columns\_number];  
 for (int i = 0; i < rows\_number; i++) {  
 String[] row = inputStream.next().split(",");  
 for (int j = 0; j < row.length; j++) {  
 data[i][j] = row[j];  
 }  
 }  
 return data;  
 }  
 */\*\* funkcja odczytująca z pliku macierz typu Integer\*/* private static int[][] readBoardInt(Scanner inputStream, int rows\_number, int columns\_number) {  
 inputStream.next();  
 int[][] data = new int[rows\_number][columns\_number];  
 for (int i = 0; i < rows\_number; i++) {  
 String[] row = inputStream.next().split(",");  
 for (int j = 0; j < row.length; j++) {  
 data[i][j] = Integer.*parseInt*(row[j]);  
 }  
 }  
 return data;  
 }  
 public static void main(String[] args) {  
 */\*\*wczytanie pliku csv z tablicami\*/* String fileName = "easy\_1.csv";  
 File file = new File(fileName);  
 try {  
 Scanner inputStream = new Scanner(file);  
// String matrix\_sizes\_str = inputStream.next();  
// String[] matrix\_sizes = matrix\_sizes\_str.split(",");  
// W jednej linijce  
 String[] matrix\_sizes = inputStream.next().split(",");  
 */\*\*odczytanie z pierwszego wiersza pliku wymiarów tablicy\*/* int rows\_number = Integer.*parseInt*(matrix\_sizes[0]);  
 int columns\_number = Integer.*parseInt*(matrix\_sizes[1]);  
//Reading arrays  
 String[][] types = *readBoardString*(inputStream, rows\_number, columns\_number);  
 int[][] values = *readBoardInt*(inputStream, rows\_number, columns\_number);  
 int[][] columns = *readBoardInt*(inputStream, rows\_number, columns\_number);  
 int[][] rows = *readBoardInt*(inputStream, rows\_number, columns\_number);  
 inputStream.close();  
  
 *print2D*(columns);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 }  
}